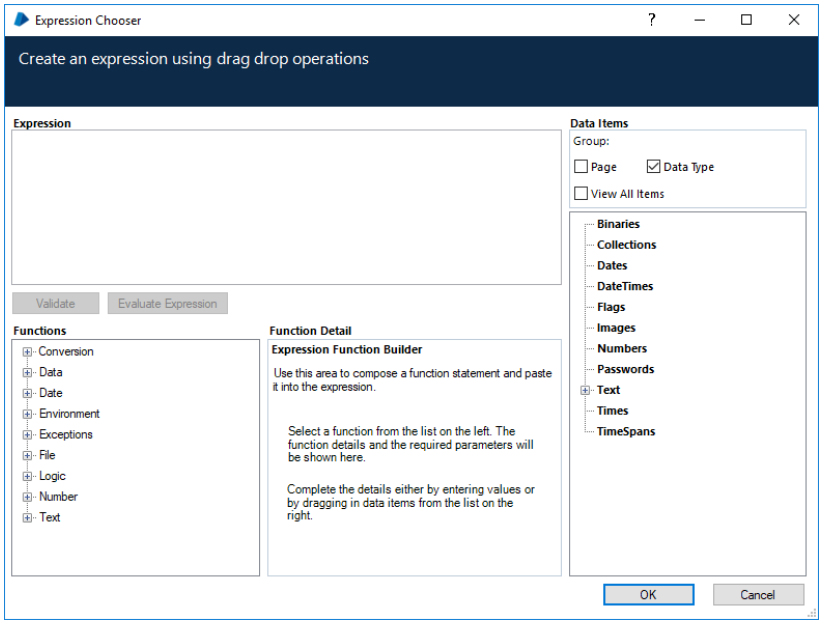
Expression Writing Guide

This guide provides a basic overview of how to write Expressions within the Stage properties of a Blue Prism Process or Business Object. You can use it to familiarize yourself with these skills before you begin the training, or to refer back to as you progress through the training.

Blue Prism Limited, 2 Cinnamon Park, Birchwood, WA2 0XP, United Kingdom
Registered in England: Reg. No. 4260035. Tel: +44 870 879 3000. Web: www.blueprism.com

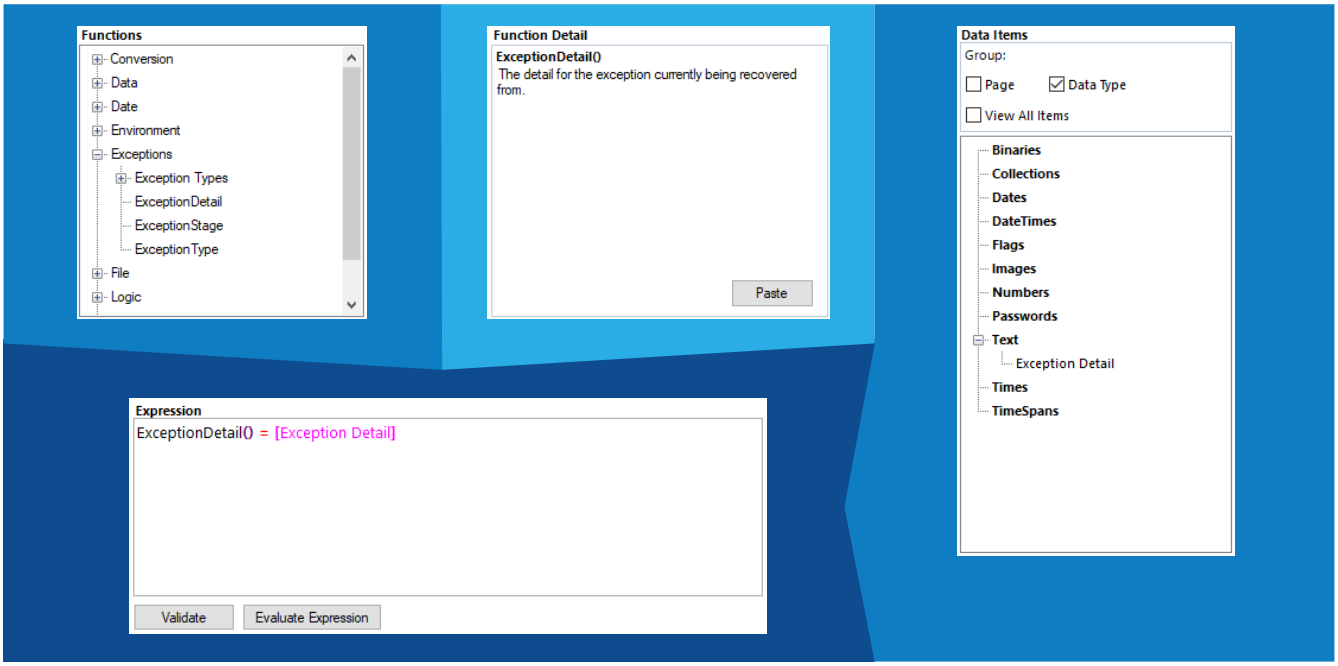
Expressions

Expressions are formulas that calculate a value, which can be used within the properties of many of the Stages within Process Studio and Object Studio. There are various features available which can assist you with the construction and testing of Expressions.



EXPRESSION CHOOSER WINDOW

The essential components for building an Expression are the Functions list, the Function Detail / Expression Function Builder area and the list of Data Items available for use within the Process Solution. The Expression area is where all of these components come together to form an Expression - which can be validated and tested using the *Validate* and *Evaluate Expressions* buttons.



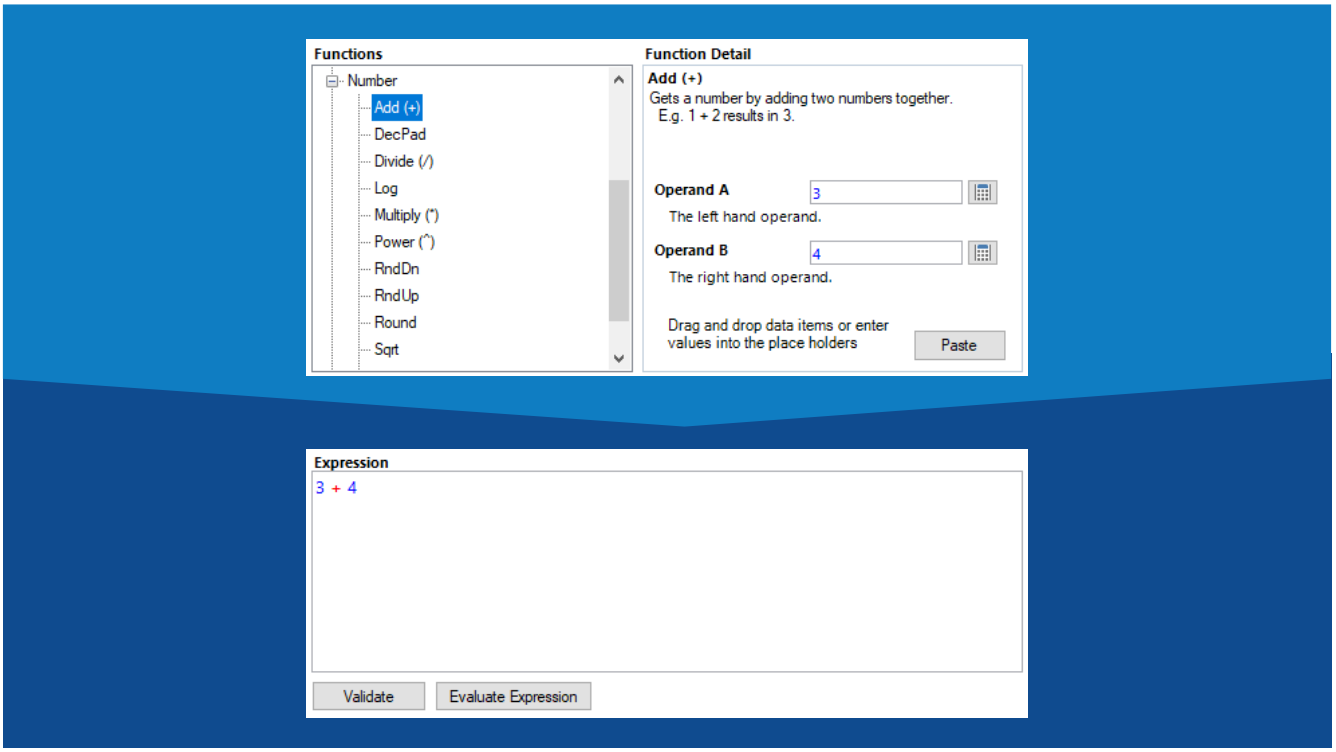
It is possible to enter Expressions using a variety of methods: users can type Expressions directly into the Expression area, drag-and-drop Functions and Data Items into the Expression area or use the Expression Function Builder to construct Functions before pasting them into the Expression area.

Functions

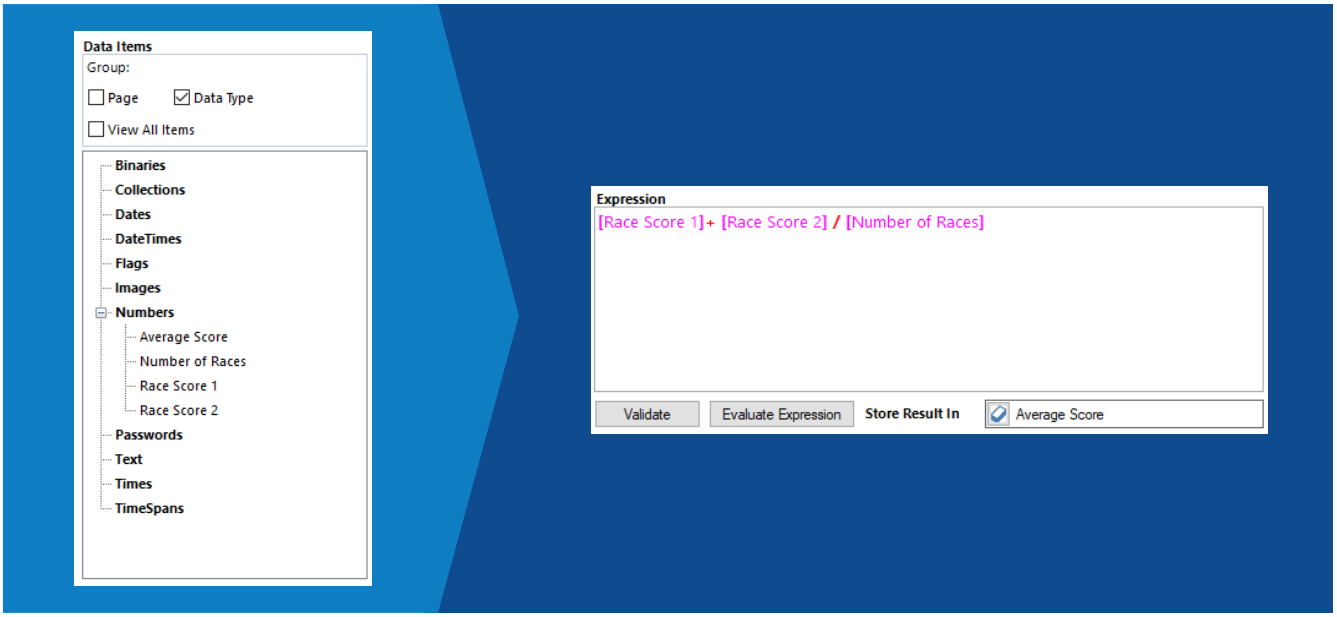
Term	Definition
Conversion	*A list of Functions used to convert values from one Data Type to another, e.g. ToDate("01/01/2020") converts a Date stored as Text into a Date Data Item*
Data	*Retrieves the number of Bytes in binary Data, to give an idea of the size of the Data*
Date	*A list of Functions that perform operations that will result in Date, DateTime, Time or TimeSpan values*
Environment	*A list of Functions that retrieve information from the Blue Prism Environment and from the Local Environment running that instance of Blue Prism, e.g. retrieving the version number of Blue Prism, BPVersionMajor() or the Windows Operating System being used, GetOSVersion()*
Exceptions	*A list of Functions that can be used to retrieve information from Exceptions, such as ExceptionDetail(), ExceptionStage() and ExceptionType(), or a literal Text value of an Exception Type that can be used to output the Exception Type as text.* ** Exception Functions can only be used between Recover and Resume Stages, when a Process or an Action is in Recovery Mode*
File	*Consists of two Functions, one used to read the contents of a file as binary Data, LoadBinaryFile(Filename), and one used to read the contents of a text file, LoadTextFile(Filename)*
Logic	*A list of Functions that use Flag values, either the literal values 'True' or 'False', or Functions that check whether a certain Text value is valid as another Data Type, producing a Flag, or Functions that perform an operation on two values to determine a Flag value, e.g. Greater than (>), Value > Value, is True; or And (AND), (Value1<Value2) AND (Value3<Value4) is True*
Number	*A list of calculation Functions that can be performed on values, such as Add (+), which simply adds two values and produces a result 1 + 2 outputs 3, or Sqrt(Number), which results in the square root value of the Number provided*
Text	*A list of Functions that can be performed on Text values, such as Trim(Text), which trims a single Text value with any white space down to just the characters, Trim(" Text ") results in "Text", or Concatenate (&), which joins two Text values together to produce a result, "Expression_" & "Guide" results in "Expression_Guide"*

Constructing Expression Statements

The Function Detail area is where an explanation of the selected Function is displayed, including a list of the fields that are required by the Function to operate. Users can enter values or drag and drop Data Items or Functions into the fields and then use the *Paste* button to add the completed Function into the Expression area.



The Data Items area is where users can select Data Items or Collection Stages for use within an Expression (by dragging and dropping) or to store the results of an Expression in the *Store Result In* field. Data Items must be wrapped in square brackets when referenced within an Expression, e.g. *[Data Item]*. When a Data Item is used within an Expression, the value or values that are stored within them will be made available for use by the selected Functions.

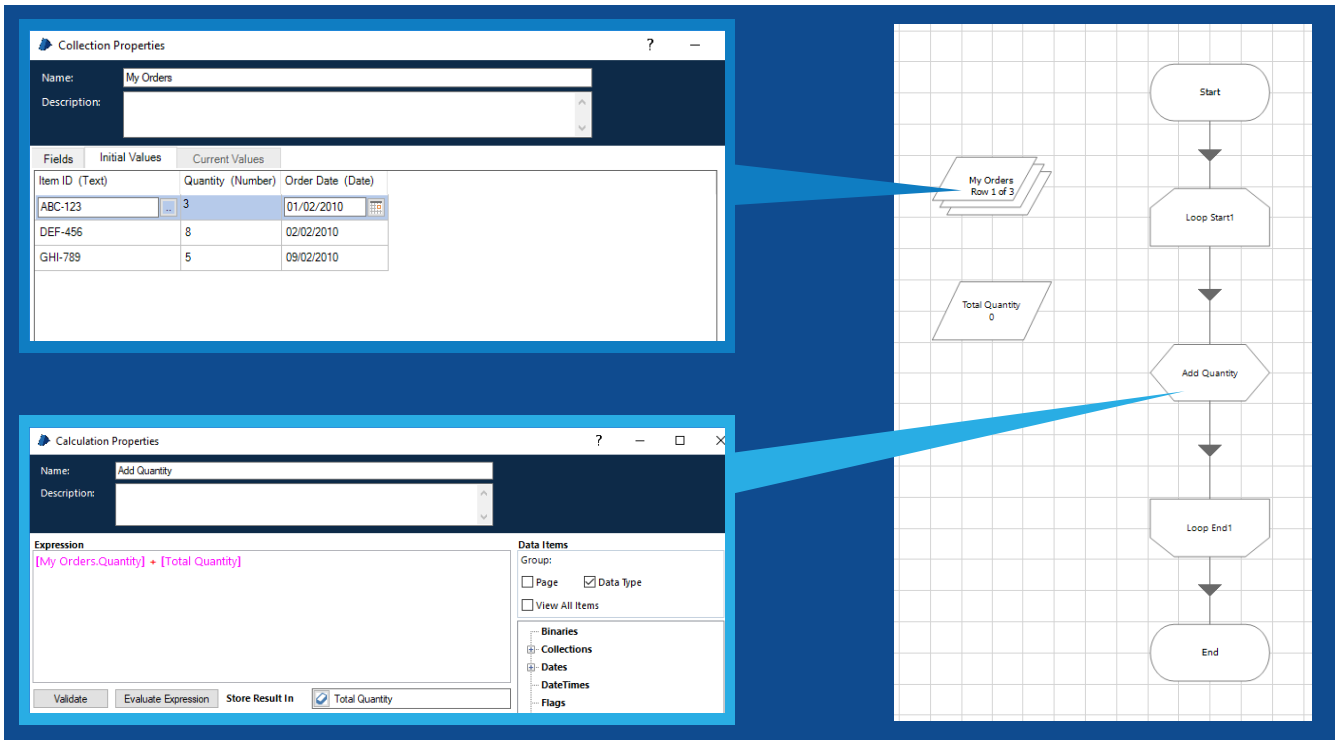


A Data Item used in the *Store Result In* field can also be used in the main body of the Expression itself.

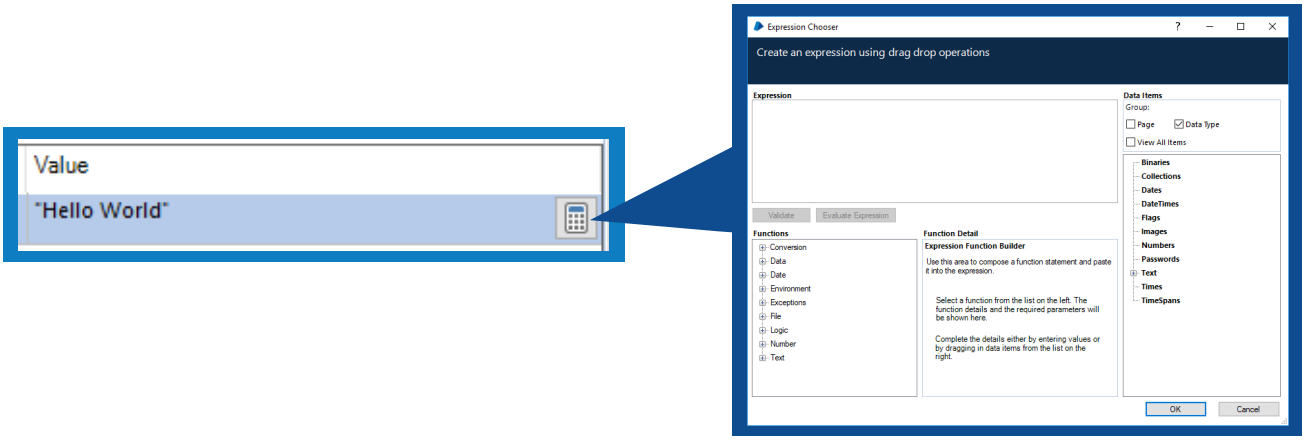
Collections store multiple values in rows and columns - to include a particular set of values (or a 'field') from a Collection within an Expression, users must reference the Collection field directly.

Collection fields can be used as part of an Expression, for example: a Calculation Stage that is positioned between the start and end of a Loop, can use the data within the Collection field to perform a calculation. As the flow passes through the Loop Start Stage, the first row of the Collection is selected and the row field values are made available for use by the Expression. As the flow passes through the Loop End Stage and loops back to the Loop Start Stage, the next row in the Collection is selected and row values made available. Once all rows within the Collection have been selected, the flow will move past the Loop End Stage.

To include a Collection field within an Expression, users must use 'dot-notation' - which is where the Collection name (e.g. *My Orders*) and the field name (e.g *Quantity*) are included within the Expression, separated by a full stop *[My Orders.Quantity]*.



The properties of some types of Stage (e.g. Decision Stages or Calculation Stages) are dedicated to building Expressions. Whereas the properties of other types of Stage contain only particular *Value* fields into which an Expression can be entered. In these cases, a small *calculator icon* to the right-hand side of the field can be clicked to launch an *Expression Chooser* window - any Expression written here, will be entered into the field upon selecting OK and closing. Alternatively, users can enter an Expression directly into the *Value* field.



Expression examples explained

Sometimes Expressions can be as simple as a Data Item, *[Full Name]*.
Other times Expressions can be more complex, *[Forename]&" "&[Middle Name]&" "&[Surname]*.
Both of these Expressions could potentially output an identical result but are formulated differently.

Expressions

[Item ID]<>""

This Expression (used within a Decision Stage in video 9.2) is made up of three parts and is asking:
is there a value in the *Item ID* Data Item?

[Item ID]	<>	""
The first part is a Data Item containing a Text value.	The second part is the Function Not equal (<>).	The third part is an empty Text value.

So the Expression states that the value contained within *Item ID* is not equal to an empty Text value, while there is a value in *Item ID* this would be True, when there is no longer a value in *Item ID* this would be False.

[Forename]&" "&[Surname]

This Expression is made up of five parts and is joining two Text values together with a space in between.

[Forename] [Surname]	&	" "
The first and fifth parts are Data Items containing Text values e.g. "*John*" and "*Smith*".	The second and fourth parts are Concatenate (&), which joins two Text values together.	The third part is a Text value with a single space between the quotation marks.

So the result would be *John Smith*.

AddDays(Today(), [Trial Duration])

This Expression is a single Function with two Parameters, *Function(Parameter, Parameter)*.

AddDays()	Today()	[Trial Duration]
The Function AddDays() that uses a Date and a Number value to calculate a future Date.	The first Parameter is a Date value, in this example the Function *Today()* is used to get the current date.	The second Parameter is a Number value, in this example obtained from a Data Item.

In this Expression the Function *AddDays()* is using todays date and adding the number of days that the *Trial Duration* lasts for, in video 4.4 the value *30* is used, so the *AddDays()* Function would calculate a Date value 30 days from today. If the value of *Trial Duration* was a negative value such as *-30*, then the *AddDays()* Function would calculate a value which was 30 days before *Today()*.

InStr(ExceptionDetail(), "Number of Races is zero") > 0

This Expression (used within a Decision Stage in video 8.4) is made up of three fundamental parts, that is asking does a certain Text value contain another Text value.

InStr()	ExceptionDetail()	"Number of Races is zero"
The first part is the Function InStr() which contains two Text value Parameters between its brackets.	The first Parameter, is used as a reference string of characters within a Text value.	The second Parameter, is compared to the first Parameter for an identical string of characters.
	>	0
	The second part of the Expression is the Function Greater than (>) which compares the size of the values either side.	The third part is the Number value zero (0).

The Expression is asking whether the string of characters that make up the Text value "*Number of Races is zero*" is present anywhere in the string of characters that make up the Text value *ExceptionDetail()*.
If the second Parameter is found within the first Parameter, then the *InStr()* Function will return a number value which represents the position in the first Parameter where the second Parameter was matched. If this value is greater than zero, then a True value will be the outcome, otherwise it will be False.

*The requirements for a Function to operate correctly can vary. Some Functions are independent and do not require any Parameters to fulfill their outcome, such as **Today()**, which simply returns today's Date value. Some Functions require one or more Parameters if they are to produce an outcome - in these cases, it is up to the user to decide which Function is appropriate and to provide the Parameters needed to produce the desired outcome.*

[Staff Data.Full-Time] = True

This Expression is made up of three parts and is asking whether the value in the *Full-Time* column in the *Staff Data* Collection is True.

[Staff Data.Full-Time]	=	True
The first part is a reference to a specific column in a Collection, dot-notation is used.	The second part is the Function Equal (=) to compare whether the values on either side are the same.	The third part is a Flag value of True.

The Expression determines whether the value contained within the *Full-Time* column in the *Staff Data* Collection is equal to True. If so, a True Flag value will be the outcome, otherwise it will be False.